福井大学 工学部 研究報告 第49巻 第2号 2001年9月
Mem. Fac. Eng. Fukui Univ., Vol. 49, No. 2 (September 2001)

A Generation Method of a Schema Manager and a Database Language Processor

Teruhisa HOCHIN*, Masahiko MATSUURA and Tatsuo TSUJI***

(Received August 24, 2001)

This paper proposes a method of generating a schema manager and a database language processor in order to support the data models and the database languages. To this end, four kinds of specifications are used. These are the structure, the database language, the semantics, and the operation specifications. Users can customize the data model and the database language by tailoring these specifications.

Key Words : Extensible Database Management System, Generator, Specification, Schema Manager, Database Language Processor

1. Introduction

In recent years, database management systems (DBMSs) have come to play an important role in advanced data intensive applications such as CAD/CAM, network management and VLSI design. These applications place a variety of demands on DBMSs, but no single DBMS can handle them all. A great deal of research has been done on the development of extensible DBMSs[1-24]. There are varieties of approaches to improving DBMS extensibility. Some approaches follow the full function DBMS approach[2, 3], and others follow the DBMS generator approach[5, 7, 10, 11, 12, 13, 24]. In the former, a full function DBMS is enhanced to have extensibility on specific functions. In the latter, a DBMS is generated or constructed with program parts. Some extensible DBMSs have the fixed DBMS kernel[2, 3, 5, 7, 11, 12], and others have no fixed DBMS kernel[10, 13, 24]. In the latter, every aspect of a DBMS is tried to be extensible. Some systems support the only one data model and the only one database language[2, 3], and others try to support one or more data models and database languages[10, 16, 17, 18, 19, 22]. *

In this paper, we focus on the extensibility on the data model and the database language. In KIDS[10] and the multi-lingual database system[19], more than one data model and more than one database language can be supported. However, language processors have to be implemented by the DBMS implementer. The work in implementing a language processor includes cumbersome tasks, e.g., making a parse tree, and checking the validity of literals by using the schema. The burden of implementing a database language processor should be decreased. On the other hand, although database languages are automatically processed in the configurable database system[18] and ORIENT[22], the data models supported are limited. Supporting any kind of data model is considered to be preferable.

This paper proposes a method of generating a schema manager and a database language processor for the purpose of supporting data models and database languages easily. The proposed method follows the specification approach[10]. *The structure specification* is for defining the structure of a data model. A database language is defined by using *the database language specification* and *the semantics specification*. *The operation specification* is for defining the operations of a data model, and their arguments. The proposed generation method is based on the fundamental data structure and the fundamental operations. The fundamental data structure is the set of records, which is called *the record set*. The fundamental operations are those of manipulating record sets. These fundamentals enable the generator to generate the schema manager and the components of a database language processor easily.

* Dept. of Information Science
** Fujitsu Hokuriku Systems Limited

This paper is organized as follows: The database language processor is briefly described in Section 2. Section 3 proposes the generation method of a schema manager and the components of a database language processor. Section 4 evaluates the proposed method in the view of productivity. In Section 5, this work is compared with related research works. Lastly, future directions are mentioned in Section 6.

2. Database Language Processor

This section briefly describes a database language processor and the managers around it. The statements of a database language are processed by a database language processor. The database language processor uses the storage manager in order to store data into a database file, and obtain data from it.

The database language processor in this paper is assumed to be composed of five components: a parser, a semantics checker, a plan generator, a query optimizer, and a plan executor as shown in Fig. 1.

The parser checks whether a statement is correct according to the grammar of the database language, and produces a parse tree.

The main role of the semantics checker is to inspect whether the literals appeared in a statement are valid. The semantics checker consults the schema manager for this check. Moreover, the semantics checker obtains the internal numbers assigned to the literals from the schema manager. For example, the table number, which is for identifying a table, is obtained for the table in a database under the relational data model.

The plan generator generates an execution plan from the parse tree that the schema information is added to by the semantics checker. Although the generated plan is correct to the given statement, efficiency of execution is not considered.

The query optimizer rewrites the execution plan into the efficient one. For this aim, the query optimizer may use the heuristics, the statistics about data, and/or the hints of data access.

The plan executor executes the database processing according to the execution plan rewritten by the query optimizer. The plan executor visits the nodes of the execution plan, and calls the appropriate functions or methods of the software library.

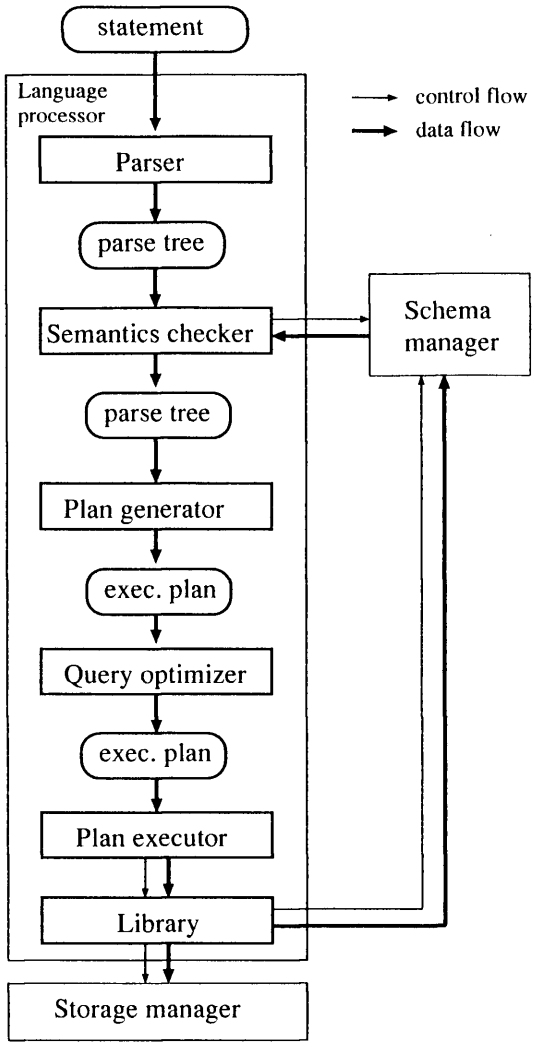

Figure 1: The architecture of a database language processor

3. Generation Method

The generator proposed in this paper generates a schema manager, a parser, a semantics checker, and a plan generator. The plan executor may be a fixed software component, i.e., the plan executor does not have to be generated. Because a plan generator generates the plans that can be executed by the plan executor. A query optimizer should be generated. One possible method is using the optimizer generator[9]. The generation of a query optimizer is beyond the scope of this paper.

For the purpose of generating those software components, the generator takes four kinds of specifications: *the structure*, *the database language*, *the semantics*, and *the operation specifications*.

First, the fundamentals for the generation method

are described. Next, four kinds of specifications are explained. After that, generation methods of a schema manager, a parser, a semantics checker, and a plan generator are described.

3.1 Fundamentals

It is very difficult to study the extensibility of all of the aspects or functionalities of the database management system. As algorithms depend on data structures, and vice versa, the algorithm is not decided unless the data structure is decided, and vice versa. In this paper, the followings are fixed: the fundamental data structure for representing objects, and the operations for manipulating them.

3.1.1 Fundamental data structure

The set of records, which is called *the record set*, is adopted as the fundamental data structure for manipulating data values. A record is composed of a set of attributes. An attribute is a triplet of its name, its data type, and a data value as in the traditional data models. In the information of an attribute, additional information may have to be considered. The information includes the flag indicating whether the NULL value is permitted, and the indicator whether the attribute is a key or a part of a key. In this paper, such information is not considered. The treatment of this kind of information is a part of future works.

Usage of the record set affects many aspects of the processing of the database language. First, the interface to the storage manager can be fixed. The storage manager handles a set of records rather than a set of values. Second is that the functions for the fundamental operations can be fixed. For example, the *projection* operation takes a record set, and produces the record set of which attributes are specified in the projection operation. If the set of values is used instead of the set of records, the function for this operation is entirely different from this function. Thirdly, the structure of the data model can automatically and easily be converted into the structure of a record set.

3.1.2 Fundamental operations

The plan executor visits the nodes of the execution plan, and calls the appropriate functions as described before. We have prepared the set of functions corresponding to the nodes of the execution plan. These are as follows, where RS denotes a record set.

- SCAN : `record_set_number`, `schema_info` → RS
  This operation obtains the record set by using the storage manager.
- LOOKUP : `oid` → `record`
  This operation obtains the record corresponding to the specified identifier (`oid`) by using the storage manager.
- SELECT : RS, `retrieval_condition` → RS
  This operation takes a record set and a retrieval condition, and produces the record set of which records satisfy the retrieval condition.
- PROJECT : RS, `set_of_project_elements` → RS
  This operation takes a record set and a set of projection elements, and produces the record set, of which records are composed of the attributes specified in the projection elements. A projection element is specified by using the INNER-NAVI or OUTER-NAVI operation.
- JOIN : RS, RS, `join_condition` → RS
  JOIN takes two record sets, and produces the record set, of which records are obtained by concatenating the records satisfying the join condition.
- UNION : RS, RS → RS
  This operation produces the union set of the records in the two record sets.
- DIFFERENCE : RS, RS → RS
  This operation produces the difference set of the records in the two record sets.
- INTERSECT : RS, RS → RS
  This operation produces the intersection set of the records in the two record sets.
- SORT : RS, `list_of_sort_spec` → RS
  This operation takes a record set and a list of sort specifications, and produces the record set, of which records are sorted by the attributes specified in the sort specifications. A sort specification is a tuple of an attribute name and the flag indicating descendant or ascendant.
- DISTINCT : RS → RS
  This operation eliminates the duplicated records in the record set.

```
%
datatype1 = All
datatype2 = Int, Double
%
name      = char[64]
datatype1 = int
datatype2 = int
column    = <name> <datatype1>
table     :: <name> <column>+ identified_by <name>
relationship : <name> <table>+ <datatype2> identified_by <name>
%
```

Figure 2: An example of the structure specification

- INNER-NAVI : `record`, `att_number` → `value`
  This operation takes a record and an attribute number as inputs, and returns the value of the attribute.

- OUTER-NAVI : `record`, `att_number`, `dest_record_set_number`, `dest_att_number` → `value`
  This operation takes a record, an attribute number, a destination record set number, and a destination attribute number as inputs. This operation works as follows. The value o for the attribute of the record is obtained as in the INNER-NAVI operation. The record d, of which an OID value is equal to o, is obtained by using LOOKUP operation. The value of the destination attribute of the record d is returned. Although this operation can be implemented by using the INNER-NAVI, the LOOKUP, and the INNER-NAVI operations, it is included in the fundamental operations because of the convenience.

- INSERT : `RS`, `record` → `bool`
  A record is inserted into the record set.

- DELETE : `RS`, `record` → `bool`
  The record is deleted from the record set.

- UPDATE : `RS`, `old_record`, `new_record` → `bool`
  The record is updated to the new one.

The SCAN operation reads all of the records of a record set sequentially. Another kind of the SCAN operation using an index is not currently supported. The attributes composing a key have to be specified for such an operation. This kind of operation will be able to be easily supported. Neither the operation of grouping records nor the operation of calculating the closure is supported yet. The JOIN operation is currently a nested-loop join. The other implementations including a merge join are not supported. Moreover, this fact requires additional study on adapting the operations of the data model level to those of the implementation level. For example, although the join operation is unique in the data model level, several implementations for this operation exist in the implementation level. If some conditions are satisfied, the more efficient function can be used. For example, the merge join can be used if the records are sorted according to the join key. Selecting the appropriate function is one of the roles of the query optimizer. This functionality is included in the future research.

3.2 Specifications

Here, four kinds of specifications are described. These are *the structure*, *the database language*, *the semantics*, and *the operation specifications*.

3.2.1 Structure specification

The structure of a data model is specified through the structure specification. A structure specification is composed of two parts: *the data type specification*, and *the main specification*. The specification of *the data type specification* is the form of

`data_type_name = data_types.`

A `data_type_name` is the literal representing the name of a set of data types. In the right hand side, data type names and/or `All` representing all of the atomic data types can be specified.

The specification of *the main specification* is the form of

`left_element { = | : | :: } right_elements.`

In this specification, `left_element` is a literal. The character "=" is used to define the structure of `left_element`. The character ":" is used to define that

```
%
[\'.(,)]     ;
"SELECT"
"FROM"
"WHERE"
"AND"
[a-zA-Z0-9_]*  IDENT
[0-9]+         NUMBER
[<>=]          OPERATOR
%
# statement    : select_stmt ;
select_stmt    : SELECT selectlist FROM fromlist WHERE condition ;
selectlist     : pair_element
               | selectlist ',' pair_element ;
pair_element   : columnname
               | tablename '.' columnname ;
pair_element2  : pair_element ;
tablename      : IDENT ;
columnname     : IDENT ;
fromlist       : tablename
               | fromlist ',' tablename ;
condition      : condition AND predicate
               | '(' condition ')'
               | predicate ;
predicate      : sel_pred | join_pred ;
sel_pred       : pair_element OPERATOR '\'' IDENT '\''
               | pair_element OPERATOR '\'' NUMBER '\'' ;
join_pred      : pair_element OPERATOR pair_element2 ;
%
```

Figure 3: An example of the database language specification

`left_element` independently exists, and that this element may have instances as well as to define the structure of `left_element`. The character "::" is used to additionally define that an instance of `left_element` has its own identifier. When an instance of `left_element` is defined to have an identifier, the literal `OID`, of which definition is `OID = int`, is automatically added to `right_elements`. Specifying this kind of implicit setting is included in future research. The `right_elements` are literals or the symbols representing the option(()), the union(|), and the repeat(+).

An example of the structure specification is shown in Fig. 2. The data type specification is specified between the first two % characters. In this example, `datatype1` represents all of the atomic data types, while `datatype2` represents the two data types: `Int` and `Double`. Next is the main specification. The literal `name` is represented with the character string, of which length is 64. The literal `datatype1`, which is already defined in the data type specification, is represented with an integer value. This means that the data type of `datatype1` is represented with an integer value, and that the data type is managed as an integer value by the schema manager. The literal `column` is represented with the concatenation of the literals `name` and `datatype1`. A `table` has instances, and the instances have their own identifiers. The literal `table` is represented with the concatenation of the literals `OID`, `name`, and one `column` or more. The literal `OID` is automatically set as described above. A `table` is identified by its `name`. The literal `relationship` is represented with the concatenation of the literals `name`, one `table` or more, and `datatype2`. Although a `relationship` has instances, they do not have their own identifiers.

3.2.2 Database language specification

The database language specification is composed of *the lexicon part* and *the grammar part*. *The lexicon part* includes the lexicons used in the database language.

This is straightforwardly converted to the lex source file. *The grammar part* includes the gram-

```
%
Logic-Operator AND OR NOT
AND-Operator = "AND"
OR-Operator  = "OR"
NOT-Operator = "NOT"
Comparison-Operator EQ LT GT
EQ-Operator  = "=" of OPERATOR
LT-Operator  = "<" of OPERATOR
GT-Operator  = ">" of OPERATOR
tablename  = <table>-<name>
columnname = <column>-<name>
%
fromlist  <distinct>
pair_element <identified_by> tablename, columnname
pair_element-tablename <correspond_to> fromlist-tablename
%
```

Figure 4: An example of the semantics specification

mar of the database language. This part includes the rules, which are specified in BNF, of the yacc source file. An example of the database language specification is shown in Fig. 3. The lexicon part is between the first two % characters. The grammar part follows the lexicon part. This is the simple and limited version of the SELECT statement in SQL. The symbol # appeared at the head of the first rule of the grammar part represents that this rule is the start rule.

3.2.3 Semantics specification

In the semantics specification, the semantics of the database language is specified.

An example of the semantics specification is shown in Fig. 4. The first line following the character % specifies that the logic operators AND, OR, and NOT are used. The next line specifies that the AND operator is represented with the character string AND. Comparison operators are also specified by using the similar specifications.

The line `tablename = <table>-<name>` is for the mapping between the language specification and the structure of the data model. This line specifies that the literal `tablename` in the language specification corresponds to the `name` of the `table` in the structure specification. The semantics checker checks whether the character string appearing as `tablename` in a statement of the database language exists as the `name` of the `table` by consulting the schema manager.

The lines following the second character % are syntactic rules rather than semantic ones. In the line `fromlist <distinct>`, it is specified that the same element does not exist in `fromlist`, which is defined in the language specification. In this example, the equality problem does not occur because the element of `fromlist` is simply a table name. The element having the same table name is the same element. However, in general, deciding whether an element is equal to another one is not trivial. For the equality test, `identified_by` is used. The line including the word `<identified_by>` is an example. A `pair_element` is distinguished from the others by using the table name and the column name. When both of the table name and the column name of an element are the same as those of another one, these two elements are judged to be identical. The last line before the last character % is for the existence check. This line specifies that the table name of a `pair_element` must appear as the table name of the element in `fromlist`.

Although these specifications are syntactic, the equality can not be able to be checked until the schema information is obtained. This is the main reason why these specifications are included in the semantics specification rather than in the language specification.

3.2.4 Operation specification

In the operation specification, two kinds of mappings are specified. The first is the mapping between the user's operations and the fundamental operations described as before. The second is the mapping from the syntax in the database language to the arguments of the fundamental operations.

```
%
Scan : SCAN
    RS = fromlist-tablename

Select : SELECT
    condition-predicate-sel_pred
        Ldata = {sel_pred-pair_element | sel_pred-NUMBER | sel_pred-IDENT }
        Rdata = {sel_pred-pair_element | sel_pred-NUMBER | sel_pred-IDENT }
        Operator = sel_pred-OPERATOR

Project : PROJECT
    Navi_list = selectlist-pair_element

Join : JOIN
    condition-predicate-join_pred
        Rdata = join_pred-pair_element
        Ldata = join_pred-pair_element2
        Operator = join_pred-OPERATOR
        RS1 = join_pred-pair_element-tablename
        RS2 = join_pred-pair_element2-tablename

Dot : INNER-NAVI
    pair_element
        RS   = tablename
        Item = columnname
    pair_element2
        RS   = tablename
        Item = columnname
%
```

Figure 5: An example of the operation specification

An example of the operation specification is shown in Fig. 5. In this example, five operations are defined. These are `Scan`, `Select`, `Project`, `Join`, and `Dot`. These operations are mapped to the fundamental operations: `SCAN`, `SELECT`, `PROJECT`, `JOIN`, and `INNER-NAVI`, respectively. The third line is the specification for the argument of the `SCAN` operation. `RS` is the abbreviation of the record set. In this line, it is specified that the table name in `fromlist` corresponds to the record set. Although the arguments of the `SCAN` operation are the record set number and the schema information as described before, the user can not specify them. Because the record set number is the internal number created by the system, and the user can not recognize it. The user can neither recognize the schema information. Therefore, names are used in the specification. The system can understand that the table name is used in order to identify the table as specified in the structure specification, and the table number, which is put to the table by the schema manager, can be obtained by using the table name in this example. This specification has another meaning. The operation `Scan` is invoked for each table in `fromlist`. For example, when two tables are specified in `fromlist`, the operation `Scan` is invoked twice for two tables. Therefore, the operation specification includes the hints to the plan generator for generating the execution plan.

Next is the specification for the operation `Select`. The next line of the line `Select : SELECT` is the path to the each predicate for the retrieval condition. In this example, a predicate is the `sel_pred` of `predicate` of `condition`. The root of the path, which is `condition` in this example, does not have to be the statement, which is `statement` in this example. This means that the operation `Select` is invoked wherever `condition` appears. This is convenient when there are many places to invoke the same operation. The following three lines specify the items of the predicate and the operator.

In the specification for the operation `Dot`, two paths are specified: `pair_element` and `pair_element2`. The operation `Dot` is invoked when `pair_element` or `pair_element2` appears.

3.3 Schema Manager Generation

The data structures for managing schema can be decided when the structure specification is obtained. The following policies are adopted in order to manage schema.

- The element that exists independently is managed as the first class citizen.
  This kind of element is defined by using : or :: in the structure specification. In the example shown in Fig. 2, `table` and `relationship` are this kind of elements. The information of this kind of element can be directly accessed. The information of the other kind of element can not be directly accessed. For example, the information of `column` can only be accessed through the information of `table`.

- The information is managed by using links.
  Consider two tables: `Employee` and `Dept`. The information on `Employee` and that on `Dept` are managed by using links rather than arrays. Moreover, in the case that the table `Employee` has the columns `name` and `address`, the information on `name` and that on `address` are linked from the information on `Employee`, and the information on `name` is linked to that on `address`.

- When the data type of the component `C` of an element `P` is not the primary one, the information on `C` is managed apart from that on `P`, and that on `C` is linked from that on `P`.
  In the example shown in Fig. 2, `table` has the components `name` and `column`. As the component `name` is the character string, the information on `name` is included in the information on `table`. On the other hand, the data type of the component `column` is not the primary one. Moreover, this component is repeatable (`<column>+`). Therefore, the information on `column` is held apart from the information on `table`.

- One or more internal numbers are put to each element, and are held in schema.
  The storage system adopts the number interface. The internal numbers for `table` and `relationship` are required in requesting the storage system to store and obtain data. The internal numbers have to be decided, and to be held in schema.

As the schema manager is the software component that manages schema, a schema manager can also be generated based on the structure specification. The generator takes a structure specification, and produces the graph representing the structure of the data model. This graph is referred to as *the data model graph*. The data model graph enables the generator to obtain the information on the structure of the data model easily.

The steps in generating the schema manager are as follows.

1. Creation of the header file
   According to the policies described above, the data structures for managing schema are decided, and are defined in the header file.
2. Creation of `create` function
   This function is for inserting the information on the element, e.g., the table `Person`, and the column `address`. In this function, the internal numbers of identifying elements are decided, and are kept in the schema.
3. Creation of `drop` function
   This function deletes the information of the specified element. This function takes one or more internal numbers identifying an element as arguments.
4. Creation of `select` function
   The information on the element can be obtained by using this function. This function also takes one or more internal numbers identifying an element as arguments.

3.4 Parser Generation

The generator generates a lex source file and a yacc source file. These files are used to generate a parser. The generator uses the database language specification and the data model graph.

The lexicon part of the database language specification can be straightforwardly converted to the lex source file. The grammar part of the database language specification includes the rules of the yacc source file. The main task of the generator is to make actions producing a parse tree, and to put them into the yacc source file.

The following observations enable us to produce a parse tree automatically.

- The parse tree for the statement in the database language is not complicated.

- Concatenations and lists are the major constructs of the statement.

The generator generates the header file as well as the lex source file and a yacc source file. In the header file, the data structures for the parse tree are defined. These data structures are decided according to the similar policies to those in deciding the data structures for managing schema. When an element is a list, the field of the pointer to the next element is included in the data structure.

The actions producing a parse tree are mainly divided into two kinds of tasks. The first is allocating the space for the appropriate data structure. The second is linking the data structure from another data structure. When an element C is a component of another element P, the data structure for C is linked from that for P. When an element C is an element of a list L, the data structure for C is linked from the previous element of L.

In the parse tree, additional spaces and links are attached. These are for the purpose of making the process of the semantics checker easy. For the elements relating schema, the spaces for the internal numbers of the elements are reserved in the corresponding data structures. Such data structures are linked one by one. Semantics checker navigates through this link in order to consult the schema manager. This link is called *the schema element link*. Moreover, all of the data structures in a parse tree are linked one by one. This link is convenient to go around all of the nodes in a parse tree.

3.5 Semantics Checker Generation

Semantics checker navigates through *the schema element link* of a parse tree to convert the names of elements into the internal numbers corresponding to them. After that, semantics checker does the distinct check and the correspondence check according to the semantics specification. The generator generates such a program source file.

3.6 Plan Generator Generation

The generator generates a plan generator. This plan generator generates a simple execution plan. That is, the performance of the execution is not considered. Converting this execution plan into the efficient one is the role of the query optimizer.

The steps generating an execution plan are currently as follows. First,the nodes corresponding to the operation SCAN are created. There may be two or more nodes of the operation SCAN. Next, the nodes corresponding to the operation JOIN are created. All of the nodes of SCAN have to be linked through the nodes of JOIN, and become one record set. When there are more than one join predicate for the same pair of the record sets, the predicate appearing first is used as the join predicate. Thirdly, the nodes corresponding to the operation SELECT are created. The whole of the retrieval condition including join predicates is set as the argument of the operation SELECT. Finally, the node corresponding to the operation PROJECT is created. Many things for creating the efficient execution plan remain to the query optimizer. Currently, only these operations are supported. Future work includes the support of all of the fundamental operations.

4. Evaluation

The generator of a database language processing system is evaluated in the view of productivity. Productivity is evaluated by comparing the number of lines of specifications and that of lines of code generated by the generator, and by comparing man-days of constructing a database language processing system with and without the generator.

In this evaluation, two database language processing systems are generated. One is that for the relational model and a part of SQL. The other is that for the functional model and a part of DAPLEX language.

The numbers of the lines of the specifications for defining the relational model and a part of SQL are shown in Table 1. The amount of the code generated by the generator is 3697 lines in C. The cost of writing the specifications is about four days. The amount of the code written from scratch is 4158 lines in C.

Table 1: Numbers of lines of specifications

Specification	Relational Model	Functional Model
Structure spec.	7	11
Database lang. spec.	64	51
Semantics spec.	14	12
Operation spec.	24	20
Total	109	94

The cost of building the database language processing system from scratch is about six months.

The numbers of the lines of the specifications for defining the functional model and a part of DAPLEX language are also shown in Table 1. The amount of the code generated is 3882 lines in C. The cost of writing the specifications is about four days. The amount of the code written from scratch is 4130 lines in C. The cost of building the database language processing system from scratch is about four months.

The ratio of the number of the lines of the specifications and that of the lines of the code generated is thirtieth. The ratio of the man-days of building the system by using the generator and from scratch is about thirtieth. Therefore, it can be said that the productivity of the generator is very high.

5. Related Works

We adopt the specification approach rather than the implementation approach. This is the same direction as in KIDS[10]. We agree with Dittrich *et al* that the specification may decrease the cost in building a DBMS. The KIDS tools will generate the major parts of a domain-specific DBMS according to the given specifications. However, the database language processor remains to be implemented by the DBMS implementer. In this paper, we tries to generate the database language processor and the schema manager. By adopting our method, it is expected that the DBMS building cost will be more decreased.

Demurjian *et al* have proposed the multi-lingual database system[19]. This system enables the data model and the database language to be incorporated on the top of the kernel data model and the kernel database language. To this end, three subsystems have to be implemented. These are the kernel mapping, the kernel controller, and the kernel formatting subsystems. The kernel mapping subsystem transforms the user's world to the kernel's world. The kernel controller subsystem asks the kernel database subsystem to process the tasks. The kernel formatting subsystem transforms the results from the kernel database subsystem to the things in the forms of the user's world. The DBMS implementer has to implement these three subsystems adding to the compiler for the user's database language. This will result in the heavy burden for the DBMS implementer.

There are several researches addressing to the support of the domain specific data models. Hong *et al* have proposed the meta model to represent the object-oriented data models[16]. Their approach is based on the specification. A variety of the object-oriented data models can be specified. Zhang *et al* have proposed the method enhancing DBMSs with complex relationship semantics in ORIENT[22]. The relationship can be customized by using a kind of specification. The modification of the relationship semantics influences the processing of the database language. Cooper has proposed the configurable data modelling system[18]. This system enables the configuration on the data model, and the user interface including command languages and graphical interfaces. Configuration is based on the form or the template. That is, the data model and the data language can be configured by selecting items from the lists and/or putting something into the forms. These three systems have tried to give the extensibility on the data models by using specifications or templates. We agree on the approaches adopted in these systems. However, there are some restrictions on the extensibility. In the approach of Hong *et al*, the target data models are limited to the object-oriented data models. In the approach of ORIENT, the extensibility is limited to the relationships. The Cooper's approach has the limitation in defining the data model and the database language. We believe that the specification is more powerful than the template.

Embury *et al* have proposed the modular compiler architecture for a functional data model[23]. This architecture makes it possible to add new compiler modules in order to construct the compilers for new sub-languages. Although it is thought that fine grained components are good for the extensibility, there are several drawbacks. First is that the target data model is fixed to the functional data model. Our approach tries to allow one or more data models. Second is that the modules have to be implemented. We think that the users would not like to write any code if possible. Our approach is based on the specifications. We believe that our approach may decrease the burden of constructing a DBMS.

6. Concluding Remarks

This paper proposed a method of generating a schema manager and the components of a database language processor. Four kinds of specifications are

used in order to specify the data model and the database language. The evaluation results show that the generator adopting the proposed method has high productivity.

There remain many issues to be addressed. In the structure specification, literals have to be defined before they are used. Therefore, recursive definitions can not be specified. Improvement of the specification capability of the structure specification is a subject for future work. A record is currently stored in a database file as it is. That is, the data format of a record in a file is the same as that of the record used by the database language processor. Many models in storing complex objects into a file have been proposed[25]. Several extensible database management systems adopt the kernel data models, which are considered to be the storage models[5, 10, 11, 12, 19, 24]. Specifying the storage model is another subject for future work. For this aim, another specification, e.g., the storage specification, may be required. The issues on the query optimization are not be addressed in this paper. One possible method is using the optimizer generator[9]. Specifying the optimization rules is another future work. Moreover, the operations in the execution level may differ from those in the data model level. That is, for example, there are several methods of implementing the join operation, while the join operation in the relational data model is unique. Although several operations are supported as the fundamental operations, additional operations will have to be added as the fundamental ones. The operations added have to be specified in the operation specification as for the original fundamental operations. The future research includes this kind of extensibility. The semantics checker currently checks the literals. However, in managing semistructured data, such checks are not required, and the information corresponding to the schema may have to be kept in inserting data[26, 27, 28]. Therefore, the behavior of the semantics checker, and the process of modifying data may have to be changeable. This kind of specification is another future work. We have studied the generation method through the limited data models and the limited database languages. We use only the relational data model and the functional data model. We will have to apply the proposed method to the other data models including the object-oriented, and object-relational ones. We use currently the subset of SQL, and the sub-set of the data manipulation language DAPLEX. The future research includes the supports of the full-sets of these languages as well as the other database languages.

References

[1] Carey, M., and Haas, L. : "Extensible Database Management Systems," SIGMOD RECORD, Vol. 19, No. 4 (1990).

[2] Stonebraker, M., Rowe, L. A., and Hirohama M. : "The implementation of POSTGRES," IEEE Trans. of Know. and Data Eng., Vol.2, No.1, pp. 125-142 (1990).

[3] Haas, L. M. *et al*: "Starburst mid-flight: As the dust clears," IEEE Trans. of Know. and Data Eng., Vol.2, No.1, pp. 143-160 (1990).

[4] Batory, D. S., Leung, T. Y., and Wise, T. E. : "Implementation concepts for an extensible data model and data language," ACM Trans. on Database Sys., Vol.13, No.3, pp. 231-262 (1988).

[5] Batory, D. S. *et al* : "GENESIS: An Extensible Database Management System," IEEE Trans. on Soft. Eng., 14, 11, pp. 1711-1730 (1990).

[6] Batory, D. S.: "Concepts for a Database system Compiler," Proc. of 7th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp.184–192 (1988).

[7] Carey, M. J. *et al* : "The Architecture of the EXODUS Extensible DBMS," Int'l Workshop on Object-Oriented Database Systems, pp. 52-65 (1986).

[8] Richardson, J. E., and Carey, M. J. : "Programming Constructs for Database System Implementation in EXODUS," Proc. of ACM SIGMOD 1987, pp. 160-171 (1987).

[9] Graefe, G., and DeWitt, D. J.: "The EXODUS Optimizer Generator," Proc. of ACM SIGMOD 1987, pp.160–171 (1987).

[10] Dittrich K. R. : "Database Technology Research at the University of Zurich: Using and Engineering Object-oriented, Active, and Heterogeneous DBMS," IEICE Tech. Report DE91-60, Vol. 91, No. 538 (1992).

[11] Scheck, H.-J., Paul, H.-B., Scholl, M. H., and Weikum, G. : "The DASDBS Project," IEEE

Trans. on Know. and Data Eng., Vol. 2, No. 1, pp. 25-43 (1990).

[12] Yaseen, R. *et al* :"An Extensible Kernel Object Management System," Proc. of OOPSLA'91, pp. 247-263 (1991).

[13] Wells, D. L., Blakeley, J. A., and Thompson C. W. : "Architecture of an Open Object-Oriented Database Management System," COMPUTER, Vol. 25, No.10, pp. 74-82 (1992).

[14] Furuse, K., Y., Yamaguti, K., Kitagawa, H., and Ohbo, N. : "Abstract Indexing Mechanism of the Extensible DBMS MODUS," Proc. of 3rd DASFAA (1993).

[15] Kojima, I. *et al* : "Design and Implementation of an Object-Oriented Database System Using an Extensible Database Toolkit," Proc. of 2nd Int'l Computer Science Conf., pp. 588-594 (1992).

[16] Hong. S. and Maryanski F. : "Representation of Object-Oriented Data Models," Inform. Sciences, No.52, pp. 247-284 (1990).

[17] Cooper, R., Atkinson, M. P., Dearle, A., and Abdrrahmane, D. : "Constructing Database Systems in a Persistent Environment", Proc. of 13th VLDB Conf., pp.117–125 (1987).

[18] Cooper, R. : "Configurable Data Modelling Systems", Entity-Relationship Approach: The Core of Conceptual Modelling (H. Kangassaro ed.), Elsevier Science Publishers B. V.(North-Holland), pp.57–73 (1991).

[19] Demurjian S. A. and Hsiao D. K. : "Towards a Better Understanding of Data Models through the Multilingual Database System," IEEE Trans. Softw. Eng., Vol. 14, No. 7, pp. 946-958 (1988).

[20] Geppert, A. and Dittrich, K. R. : "Constructing the Next 100 Database Management Systems: Like the Handyman or Like the engineer?," ACM SIGMOD RECORD, Vol. 23, No. 1, pp. 27–33 (1994).

[21] Orlandic, R. : "Foundations of a Methodology of DBMS Decoupling for Evolutionary Component DBMS Design," Proc. of 1998 International Database Engineeing and Applications Symposium (IDEAS'98), pp. 178–187 (1998).

[22] Zhang, N., Härder, T. : "On a Buzzword "Extensibility" – What We Have Learned from the ORIENT Project?," Proc. of 1999 International Database Engineeing and Applications Symposium (IDEAS'99), pp. 360–369 (1999).

[23] Embury, M. Suzanne, and Gray, M. D. Peter : "A Modular Compiler Architecture for a Data Manipulation Language," Proc. of 14th British National Conf. on Databases (BNCOD14), pp. 170–188 (1996).

[24] Hochin, T. and Inoue, U. : "An Extensible DBMS Composed of Specific DBMSs," Proc. of International Symposium on Next Generation Database Systems and Their Applications, pp.180–187 (1993).

[25] Valduriez, P., Khoshafian, S. N., and Copeland, G. P. : "Implementation Techniques of Complex Objects," Proc. of 12th VLDB, pp. 101-110 (1986).

[26] Goldman, R. and Widom, J. : "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases," Proc. of the 23rd VLDB Conf., pp. 436–445 (1997).

[27] Nakata, M., Hochin, T., and Tsuji, T. : "Bottom-up Scientific Databases Based on Sets and Their Top-down Usage," Proc. of International Database Engineering & Applications Symposium (IDEAS'97), pp. 171–179 (1997).

[28] Hochin, T., Nakata, M., and Tsuji, T. : "A Flexible Kernel Data Model for Bottom-Up Databases and Management of Relationships," Proc. of the 1998 International Database & Engineering Applications Symposium (IDEAS'98), pp. 170–177 (1998).